CS 710: Complexity Theory 10/4/2011

Lecture 8: Alternatation

Instructor: Dieter van Melkebeek Scribe: Sachin Ravi

In this lecture, we continue with our discussion of the polynomial hierarchy complexity class PH. We discuss three alternate (but equal) characterizations of the polynomial hierarchy class, one of which involves the introduction of a new model of computation called alternating Turing machine (ATM), which are a generalization of the nondeterministic Turing machine model. As an application of the notion of alternation, we discuss time-space lower bounds for SAT.

1 Alternate Characterizations of the Polynomial Hierarchy

1.1 Using oracle Turing machines

Claim 1. $\forall k \geq 0, \Sigma_{k+1}^p = \mathrm{NP}^{\Sigma_k^p}$.

Proof. For $k = 0$, the statement reduces to $\mathrm{NP} = \mathrm{NP}^{\mathrm{P}}$. $\mathrm{NP} \subseteq \mathrm{NP}^{\mathrm{P}}$ because we can choose to derive the solution non-deterministically and can ignore the oracle. $\mathrm{NP}^{\mathrm{P}} \subseteq \mathrm{NP}$ because any computation done by the oracle can also be done nondeterministically without the help of the oracle.

For $k \geq 1$, we will prove the case $k = 1$ and the same idea used can be repeated for higher values of k. For $k = 1$, the statement reduces to $\Sigma_2^p = \mathrm{NP}^{\mathrm{NP}}$ Suppose we have some $L \in \Sigma_2^p$. Then,

$$x \in L \iff ((\exists y_1 \in \Sigma^{|x|^c})(\forall y_2 \in \Sigma^{|x|^c}))\big[\langle x, y_1, y_2\rangle \in V\big]$$

We can have a non-deterministic TM M guess the value of y_1 in above formulation so that all that is left to solve is $L' = (\forall y_2 \in \Sigma^{|x|^c}))\big[\langle x, y_1, y_2\rangle \in V\big]$. Since the previous is exactly the class Π_1^p and $\Pi_1^p = \mathrm{coNP}$, we can use the oracle to test whether $< x, y > \in \overline{L'}$ and use the opposite value of the output of the oracle. Thus, $L \in \mathrm{NP}^{\mathrm{NP}}$.

Suppose $L \in \mathrm{NP}^{\mathrm{NP}}$. Let $L = L(M^{\mathrm{L}'})$ where M is a nondeterministic Turing machine and L' has verifier $V' \in \mathrm{P}$ since $L' \in \mathrm{NP}$. We can generate Σ_2^p to decide L as follows:

1. Existential Phase
    - We will use the existential operator to express a computation path of M on input x, including all queries M makes to oracle L' and the answers M receives in return.
    - The existential operator will also ensure that all positive query responses are valid by expressing a witness w_i for verifier V' for all queries expected to receive a positive response from L'.
2. Universal Phase
    - We use the universal operator to check that queries guessed to receive a negative response from L' are valid by checking that there exist no witnesses for these queries.
3. Predicate

- The predicate ensures that the computation path expressed is an accepting computation path for M.
- Additionally, the predicate checks that $V'(q_i, w_i) = 1$ for all queries q_i which are slated to receive a positive response from L.
- Lastly, the predicate checks that $\neg V'(q_i, w_i) = 1$ for all queries q_i which are slated to receive a negative response from L'.

The formula constructed in this fashion decides L, meaning $L \in \Sigma_2^p$.

$\square$

Claim 2. $\forall k \geq 0, \Pi_{k+1}^p = \mathrm{coNP}^{\Pi_k^p}$.

Proof. Follows from previous proof by complementation. $\square$

1.2 Using Boolean Circuits

Definition 1. *(Boolean Circuits)*

An n-input, single-output Boolean Circuit is a directed acyclic graph with n sources and one sink. All inner nodes of this graph are called gates and are labeled $\wedge$, $\vee$, or $\neg$ (corresponding to AND, OR, and NOT gates). The leaf-nodes of this graph correspond to variables.

Claim 3. *A Language $L \in \Sigma_k^p$ can be expressed as an exponential size boolean circuit, with k+1 alternating levels of AND and OR gates, with an OR gate at the top-most level, with the fan-in of the bottom level being polynomial bounded, and such that each bit in the description of circuit can be computed in polynomial time.*

Proof. Suppose language $L \in \Sigma_k^p$. We create a Boolean Circuit to decide L as follows: We create a circuit evaluating V for all possible combinations of possible values for $y_1, ..., y_k$ and a specific value for x. Thus, for each level i, the level will contain an array of AND or OR (depending on whether y_i corresponds to an universal or existential quantifier) gates to enumerate all possible combinations of $y_1, ..., y_{i-1}$. Each of these k levels will have about $2^{c \cdot n}$ branches coming out to the next level. We can express $V(y_1, ..., y_k, x)$ specifically as a CNF or DNF formula depending on the value of k. If k is odd (meaning that the k^{th} level has an array of OR gates), we have the formula be DNF and vice-versa. We do this because instead of having $k+2$ levels after including the CNF or DNF formula in the circuit, we can merge the $(k+1)^{th}$ level with the k^{th} level since they will have the same type of gate. Thus, we get a Boolean Circuit with $(k+1)$ levels. All the branches of the k^{th} level decide a specific value of x because by the k^{th} level, all the values of the y's have been set. Thus, an exponential number of branches come out of the k^{th} level, representing every possible value of x. And, only n branches come out of the $(k+1)^{th}$ level (representing each bit of x for each specific guess for x), assuming that x has length n, meaning that the fan-out of the last level is polynomially bounded. For each of these guesses of x, we evalute V, which we can do in polynomial time, and hardcode the value of V into the $(k+1)^{th}$ level as a branch and this determines what the circuit outputs for that value of x. Clearly, this construction of the circuit is of exponential size. In order to calculate a bit of the circuit in polynomial time, we only need to associate the bit of the description with a level of the circuit in order to know what gate it represents in order to learn its value. This calculation can be done in polynomial time.

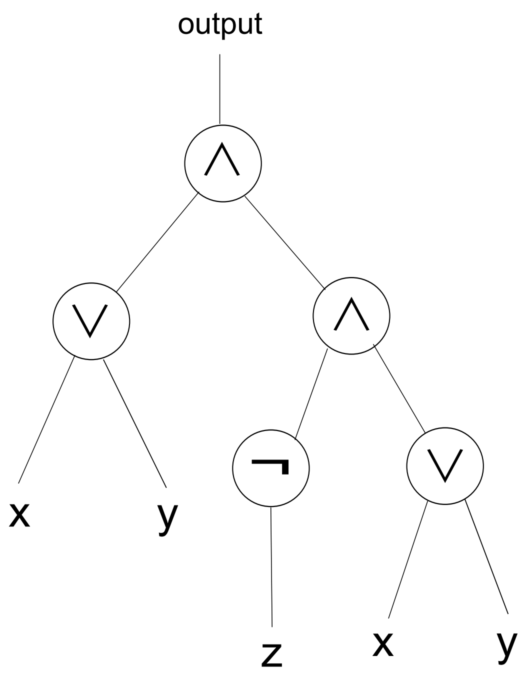

Figure 1: A boolean circuit representing $f(x, y, z) = (x \vee y) \wedge (\overline{z} \wedge (x \vee y))$

Consider a boolean circuit as defined above. We aim to construct a Σ_k^p formula which can compute the languade decided by the boolean circuit. We begin with the OR gate G_1 at the top of the circuit. The OR gate's output will be 1 if any of the gates from a lower level (which will be set of AND gates) ouptut 1. This condition can be expressed as

$$(\exists G_2)(G_2 \text{ outputs } 1) \tag{1}$$

where G_2 is an AND gate from a lower level. Similarly, G_2's output will be 1 if all the OR gates from a lower level output a 1. Meaning (1) is equivalent to:

$$(\exists G_2)(\forall G_3)(G_3 \text{ outputs } 1) \tag{2}$$

where G_3 is any gate from lower level that is connected to G_2. In this manner by repeatedly applying these steps, we can build the formula for Σ_k^p with the final formula representing the output being produced at the $(k+1)^{th}$ level. The circuit is of exponential size, meaning we can represent each of the gates in the circuit using polynomial sized indices. We can evaluate the last predicate of the formula in polynomial time because for each gate (given the index), we can identify the sub-level gates that need to be evaluated and the bits from the input that go into that gate in polynomial time since each bit of the circuit description is computable in polynomial time (by condition on the boolean circuit).

□

1.3 By Alternating Turing Machines

Alternating Turing Machines (ATMs) are a generalization of nondeterministic Turing machines. Like NTMs, though ATMs are not a realistic computational model, studying them is helpful because they capture an important class of languages. Whereas NTMs represent the class NP, ATMs are used to represent the class PH.

ATMs are similar to NDTMs in that their transition function is a relation; however, for NDTMs, every internal state (excluding q_{accept} and q_{reject}) is labeled with either the existential($\exists$) or universal($\forall$) operator and has different conditions for being an accepting configuration based on this labeling. We say that a configuration C_i of an ATM is accepting if:

1. C_i is a halting configuration during which the machine is in an accept state.
2. C_i has state labeled with existential operator and *at least one* configuration reachable from C_i in one step is an accepting configuration.
3. C_i has state labeled with universal operator and *all* configurations reachable from C_i in one step are accepting configurations.

An input x is accepted by a ATM machine if the intial configuration is accepting. In order to prevent infinite cycling, we can use a timing mechanism (bounding the computation by the total number of possible configurations) to stop the machine and reject. Now we will define classes of the polynomial hierarchy in terms of ATMs.

Claim 4. $\Sigma_k^p = \{L|L$ *is accepted by ATM with existential start state running in polynomial time and having at most* $k-1$ *quantifier alterations*$\}$

Proof. Suppose $L \in \Sigma_k^p$. We can build a ATM M with k stages where the i^{th} stage will guess the value of y_i. To have the qualifiers of y_i's match up, all the states in i^{th} stage must be labeled with the existential operator if k is odd and with the universal operator if k is even. For simulation of the predicate V, we do not need nondeterminism so we only need to label the states involved in the simulation of V such that the alternating structure of the machine (in terms of the labeling of the states) is maintained. So, M is an ATM that recognizes L and does not have more than $k-1$ qualifier alternations because we have constrained M to only use k stages with alternations occuring between each stage.

Consider a ATM M that has at most $k-1$ alternations, with an existential start state, and $L' = L(M)$. We construct a Σ_k^p formula to capture L' as follows. Since by defintion M halts in polynomial time, it does not spend more than polynomial time in any of its alternating stages. We model the choice made by M in i^{th} stage as a polynomial length string y_i. This string is quantified with the existential operator if states in the i^{th} stage have an existential operator label or with the universal operator if the states have an universal operator label. The verifier V is built to check that the nondeterministic choices made are valid for the input and that the final state reached by M is halting and accepting. □

Claim 5. $\Pi_k^p = \{L|L$ *is accepted by ATM with universal start state running in polynomial time and having at most* $k-1$ *quantifier alterations*$\}$

Proof. The same proof as above can be used as the variation in the labeling of the start state produces the intended result. □

2 Time and Space Results involving ATMs

We define ATIME($t(n)$) as the set of all problems that can be solved in time $t(n)$ on an ATM with unconstrained number of alterations for input of size n. Similarly, ASPACE($s(n)$) is the set of all problems solvable in space $s(n)$ on an ATM with an unconstrained number of alterations for inputs of size n.

Theorem 1. *NSPACE*$(s(n)) \subseteq$ *ATIME*$((s^2(n))$

Proof. The proof for this theorem follows closely our previous proof for NSPACE($s(n)$) $\subseteq$ DSPACE($s^2(n)$). In that proof, we constructed similar to a Σ_k^p formula using a divide-and-conquer strategy. The existential qualifier was used to guess the intermediate configuration (of length $O(s(n))$) of the nondeterministic Turing machine, while the universal qualifier was used to enforce the condition that we had to go from the first configuration to the middle configuration and from the middle configuration to the halting configuration. There were a total of $O(s(n))$ such qualifiers in the formula for entire process. Additionally, the predicate verified whether it was possible to go from one configuration to another and takes $O(s(n))$ time to do this since each configuration is of $O(s(n))$ size. Since the guessing step involves $O(s^2(n))$ time, it is clear that the ATM can simulate the NTM in $O(s^2(n))$ time. □

Theorem 2. *ATIME* $(t(n)) \subseteq$ *DSPACE*$(t(n))$

Proof. We will simulate the $O(t(n))$ ATM M using a deterministic TM S using $O(t(n))$ space. Basically, what we have to do is try all computation paths for M. The naive way to do this would be to do a depth-first search of M's computation tree, storing each configuration of M along a certain path. Since each configuration takes up $O(t(n))$ space and a certain path can be at most $O(t(n))$long, this would take $O(t^2(n))$ space. We do better by observing that we do not need to store the entire configuration but only need to record the nondeterministc choice that we made at each step (which can be done in a constant number of bits). TM S will accept if it determines through this seach that the starting configuration is indeed an accepting configuration. Thus, using this method we only take up $O(t(n))$ in our machine S when simulating M. □

Theorem 3. $ASPACE(s(n)) = \bigcup_{c>0} DTIME(2^{c \cdot s(n)})$

Proving this theorem will be a homework exercise.

Corollary 1. $AP = PSPACE$

Corollary 2. $AL = P$

Corollary 3. $APSPACE = EXP$

3 Application: Time-Space Lower Bounds for SAT

Though it is generally hypothesized that it takes time exponential in the number of variables to solve the SAT problem, it has not been proven that it cannot be done in linear time. Similarly, though it is also hypothesized that it takes linear space to solve SAT, it has not been proven that SAT cannot be solved in logarithmic space. However, if we take time and space into consideration together, we can derive some lower bounds for time and space usage for SAT.

Definition 2. *DTISP(t,s) is the class of languages decided by a deterministic TM in time $t(n)$ and space $s(n)$, letting n be the size of the input.*

Note that DTISP(t,s) is not the same as the set DTIME$(t) \bigcap$ DSPACE(s). DTIME$(t) \bigcap$ DSPACE(s) contains problems which are solvable in time t (with no restriction on space) and with space s (with no restriction on time) whereas DTISP(t,s) contains problems which are solvable with time t and space s.

The next lemma shows that if the space used by a deterministic TM is small enough, we get a "speed up" by simulating this machine on a ATM. We use this lemma to prove a later theorem that will establish time-space lower bounds for SAT.

Lemma 1. *DTISP$(t,s) \subseteq \Sigma_2$ TIME$(\sqrt{t \cdot s})$*

Proof. Suppose M is a TM that runs in the bound stated above and accepts a language $L \in$ DTISP(t,s). Without loss of generality, suppose M has unique accepting configuration c_l. Let input x be of length n. The computation tableau for M will have $O(t(n))$ rows and $O(s(n))$ columns. Letting c_0 be initial configuration, it is clear that $x \in L \iff c_0 \vdash_{M,x}^{t(n)} c_l$. Similar to the proof of NSPACE$(s) \subseteq$ DSPACE$(s^2(n))$, we split up the computation tableau but rather than break it up into two parts (as we did previously), we break the tableau up into b parts, where b's value is to be determined. Thus, we proceed by breaking up the tableau into b configurations $c^1, ..., c^{(b-1)}$ where $c^{(b)} = c^l$ and by verifying that for each $0 \leq i < b$, $c_i \vdash_{M,x}^{t(n)} c_{i+1}$. This calculation reduces to the condition:

$$x \in L \iff (\exists c^{(1)}, \ldots, c^{(b-1)})(\forall 0 \leq i < b) \left(c^{(i)} \vdash_{M,x}^{t(n)/b} c^{(i+1)} \right)$$

where $c^{(0)} = c_0$ and $c^{(b)} = c_l$.

Let us now analyze the time required for an ATM to perform this computation. This is a Σ_2^p calculation since each of c_i is of O$(s(n))$ space and the calculation of $c^{(i)} \vdash_{M,x}^{t(n)/b} c^{(i+1)}$ takes $O(t(n))$ time. The machine takes $b \cdot s$ time for the existential guess, as there are b blocks and s bits. It takes $log(b)$ time for the universal guess. Lastly, it takes t/b time to do the predicate computation. Thus, the total running time for the machine is $O(b \cdot s + log(b) + t/b)$. The optimum size for b is then defined by:

$$b \cdot s = t/b$$

Meaning, $b = \sqrt{t/s}$. Thus, the running time for the ATM is $O(\sqrt{ts})$.

□

4 Next Time

We will continue with our discussion of Time-Space Lower Bounds for SAT and will later discuss nonuniform models of computation.

Acknowledgements

In writing the notes for this lecture, I perused the notes by Mushfeq Khan and Chi Man Liu for lecture 7 from the Spring 2010 offering of CS 710, and the notes by Ashutosh Kumar for lecture 6 from the Spring 2010 offering of CS 710.